Incremental Learning of NCM Forests for Large-Scale Image Classification

Marko Ristin[1] Matthieu Guillaumin[1] Juergen Gall[2] Luc Van Gool[1,3]
[1]ETH Zurich [2]University of Bonn [3]KU Leuven

Abstract

In recent years, large image data sets such as "ImageNet", "TinyImages" or ever-growing social networks like "Flickr" have emerged, posing new challenges to image classification that were not apparent in smaller image sets. In particular, the efficient handling of dynamically growing data sets, where not only the amount of training images, but also the number of classes increases over time, is a relatively unexplored problem. To remedy this, we introduce Nearest Class Mean Forests *(NCMF), a variant of Random Forests where the decision nodes are based on nearest class mean (NCM) classification. NCMFs not only outperform conventional random forests, but are also well suited for integrating new classes. To this end, we propose and compare several approaches to incorporate data from new classes, so as to seamlessly extend the previously trained forest instead of re-training them from scratch. In our experiments, we show that NCMFs trained on small data sets with 10 classes can be extended to large data sets with 1000 classes without significant loss of accuracy compared to training from scratch on the full data.*

1. Introduction

The advent of large data sets such as "ImageNet" [7] or "80 Million Tiny Images" [23] has introduced new challenges compared to earlier data sets with far fewer classes [10]. As the number of classes grows and their semantic and visual distances shrink, conventional one-vs-all classifiers become tedious to train and are outperformed by nearest neighbour or multiclass approaches [1, 6, 14, 26].

A common hypothesis for multi-class learning is that the number of classes is known when the training phase starts. This implies that the aforementioned large datasets are typically conceived as *static*: the choice of classes is fixed, classifiers are trained for those, and any modification to the list of classes (*e.g.*, adding a new class) implies to retrain all the models from scratch. Considering the ongoing explosion of visual data in our daily lives and the dynamics of social networks, this is unlikely to be a realistic real-life scenario: new visual classes are bound to emerge and will eventually justify their recognition by vision algorithms.

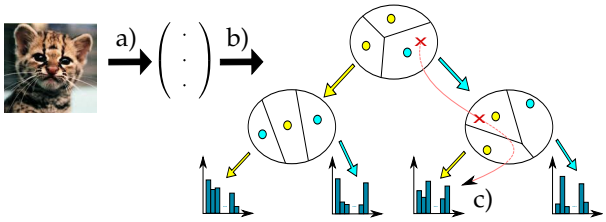


Figure 1: Classification of an image (illustrated by the red cross) by a single tree. (a) The feature vector is extracted, (b) the image is assigned to the closest centroid (colors indicate further direction), (c) the image is assigned the class probability found at the leaf.

In this light, the static learning scheme which is prevalent today seems particularly unfit. Indeed, not only would one want to prevent re-training the entire system, but learning to recognize a single new class should be much faster when many classes are already known. We can thus observe, in the context of large-scale classification, a nascent interest in *dynamic* learning scenarios [14].

In this paper, we consider the large-scale multi-class learning scenario where new classes gradually become available. Rather than having to re-train our classifier from scratch at each step (such as would be necessary for, *e.g.*, multiclass SVMs [1]), we aim at a system that can gracefully integrate new classes while limiting the loss of accuracy compared to the same system trained from all classes jointly. We refer to this problem as *incremental learning*.

As a baseline for incremental learning, one could for instance consider training a new one-vs-all classifier for each additional class. Not only does this come with a large computational burden for each new class [6], but, ultimately, the previously trained classifiers will also need to be updated to improve their performance.

To address these challenges, the first contribution in this paper is a new type of random forests [5]. Inspired by Nearest Class Mean (NCM) classifiers [14], the decisions at each node are based on the Voronoi cells formed by a small random subset of the class means observed at that node, the *centroids*. We illustrate the concept in Fig. 1. The centroids partition the feature space and assign a sample either to the left or the right subtree. We name these forests *Nearest*

Class Mean Forests (NCMF). Our experiments show that such forests outperform conventional random forests and match state-of-the-art on the challenging large-scale ImageNet dataset [3].

As a matter of fact, Random Forests are very good candidates for incremental learning. First, they are naturally multi-class and simply updating the class statistics of the leaves is already a reasonable way of adding new classes. Second, thanks to their hierarchical structure, deeper modifications to the forest can be made locally, and therefore impact only a fraction of the data at a fraction of the computational cost. The second contribution of this work is the introduction of efficient approaches for updating the forest structure in order to integrate new classes so as to maintain high accuracy at the lowest possible cost. We present an extensive experimental comparison of those approaches, studying the influence of all parameters (including the number of initial classes) on the classification accuracy and on the computational cost of training and testing the NCMF.

2. Related work

Image classification on large data sets has been widely studied [6, 13, 20], also exploiting Random Forests [5, 28]. In [28], the authors use strong SVM classification for the splitting nodes, leading to a very powerful method able to address fine-grained classification.

To address the problem of large data, there has been a wide use of *online learning* methods, such as stochastic gradient descent [13, 20]. These methods can learn from one data instance at a time, hence remain frugal in terms of memory. The assumption of online learning is that samples are provided in a uniformly random sequence, but the number of classes and the class labels are known beforehand.

Online learning has also been explored in the context of Random Forests. This is typically done by extending the trees as more samples are available. The approach of [11] first trains the trees in an extremely random fashion and subsequent updates are performed only at the leaves. [9, 18, 25] lazily split the nodes or destroy the (sub)tree after a sufficient number of samples have been observed or a certain threshold on information gain has been passed. In [27], a Hough Forest is trained incrementally with user feedback, in an *active learning* scenario. Leaves are refined at each step. Like most existing classifiers (*e.g.*, SVM or [28]) or other active or online learning methods, these works do not consider observing new classes in the data stream, and are typically not straightforward to adapt to this scenario.

Incremental learning, as we define it in this paper, is precisely interested in data streams where classes are provided in sequence. Typically, a few classes are available to start with, and new classes are added thereafter. In that vein, the approach of [14] consists in learning a discriminative metric on the initial set of classes, and classifies samples simply based on the nearest class mean. Adding a new class consists in inserting its mean in the pool of classes, leading to a near-zero cost. However, the metric itself is not updated as new classes appear, which will ultimately lead to suboptimal performance. Instead, we propose to update the structure of our forests to integrate new classes.

Incremental learning is also related to transfer learning, where the goal is to reduce the amount of labelled data required to learn a new class. It can be implemented via additional SVM constraints [22], regularizers [2], transferred features [21], shared deformable parts [16] or iteratively learned object localization based on global annotations [12]. It is also possible to exploit given or computed class hierarchies [8, 17, 19]. This is relevant here given the implicit class hierarchy that random trees provide. We differ from these approaches in that we want to add a new class efficiently rather than trying to exploit the knowledge from previous classes to reduce the amount of annotation necessary for good performance. Moreover, transfer learning is typically restricted to one-vs-all classification.

The structure of our random forests is similar to that of vocabulary trees [15] and its online kin [29], which are unsupervised clustering forests. However, where vocabulary trees use the k-means algorithm to learn the splitting nodes, we exploit class information. Following [14], the simple use of class means as centroids makes it extremely efficient to train our forests, and results in discriminative leaves and state-of-the-art performance.

3. NCM Forests

Before introducing the concept of Nearest Class Mean Forests (NCMFs), we briefly describe image classification based on NCM, which has been used for large-scale image classification in [14], and Random Forests (RF), which are commonly used for image classification, *e.g.*, in [4].

3.1. Nearest Class Mean classifier

With an image I being represented by a d-dimensional feature vector $\vec{x} \in \mathbb{R}^d$, we first compute the class centroid c_κ for each class $\kappa \in \mathcal{K}$:

$$c_\kappa = \frac{1}{|\mathcal{I}_\kappa|} \sum_{i \in \mathcal{I}_\kappa} \vec{x}_i, \tag{1}$$

where $\mathcal{I}_\kappa$ is the set of images labeled with class κ. Since there is a centroid for each class, the set of centroids $\mathcal{C} = \{c_\kappa\}$ has cardinality $|\mathcal{C}| = |\mathcal{K}|$.

Nearest Class Mean (NCM) classification of an image I is then formulated as searching for the closest centroid in feature space:

$$\kappa^*(I) = \underset{\kappa \in \mathcal{K}}{\operatorname{argmin}} \left\| \vec{x} - c_\kappa \right\|^2, \tag{2}$$

where $\vec{x}$ is the feature vector of I. Without additional refinements, the classification of one image implies $|\mathcal{K}|$ comparisons in $\mathbb{R}^d$. To improve the classification accuracy and speed, [14] replaces the Euclidean distance in Eq. (2) with a low-rank Mahalanobis distance optimized on training data.

3.2. Random Forests

Random forests [5] consist of T independently trained decision trees. At each node n of each tree, the training data S^n arriving at that node is divided by a *splitting function* $f^n : \vec{x} \mapsto \{0,1\}$ into two subsets $S^n_{f^n=0}$ and $S^n_{f^n=1}$. Commonly used splitting functions are axis-aligned [5] or linear splitting functions [4]. For training, a random set of splitting functions $\mathcal{F}^n$ is generated and the best one, f^n, is selected according to the information gain U:

$$\begin{aligned} U(f) &= H\left(S^n\right) - \sum_{i\in\{0,1\}} \frac{|S^n_{f=i}|}{|S^n|} H(S^n_{f=i}) \\ H(S^n) &= -\sum_{\kappa\in\mathcal{K}} P(\kappa|S^n) \ln P(\kappa|S^n) \\ f^n &= \operatorname*{argmax}_{f\in\mathcal{F}^n} U(f) \end{aligned} \tag{3}$$

where H denotes class entropy and $P(\kappa|S^n)$ the fraction of S^n belonging to the class κ. The left and right children nodes are then trained on $S^n_{f^n=0}$ and $S^n_{f^n=1}$, respectively, and the training continues recursively.

Given a pre-defined constant μ, the splitting stops when no $f \in \mathcal{F}^n$ satisfies $\left|S^n_{f=0}\right| > \mu$ and $\left|S^n_{f=1}\right| > \mu$. At each leaf node l of a tree t, we store the distribution over classes observed during the training, *i.e.*, $P^t_l(\kappa)$. For classification, the feature vector of the image is extracted and passed through each tree until it arrives at leaf $l(\vec{x})$. The class probabilities of all trees are averaged and classification is defined by:

$$\kappa^*(\vec{x}) = \operatorname*{argmax}_{\kappa} \frac{1}{T} \sum_t P^t_{l(\vec{x})}(\kappa)\,. \tag{4}$$

3.3. Combining NCM and Random Forests

Random Forests are efficient to train since each tree and each node at the same depth can be trained independently. Their performance heavily depends on the chosen splitting functions. In this section, we propose to use a variation of NCM classifiers as splitting functions, and we name the resulting forests NCM Forests. NCM classifiers are modified in two aspects. First, at any particular node, only a fraction of the classes will be used, hence speeding up Eq. (2). Second, the multiclass output of NCM is translated into a binary output (left vs. right child) by assigning the classes to either side.

The benefit of such an NCM Forest compared to NCM classification is that only a few comparisons are required at each node, implictly encoding a hierarchical structure of classes. This results in state-of-the-art accuracy without resorting to expensive metric learning. Compared to the most common variants of Random Forests, NCM Forests also offer non-linear classification at the node level.

More specifically, we perform the following procedure to train a node n with the data S^n arriving at that node. First, we denote by $\mathcal{K}^n$ a random subset of the classes observed in S^n, and by S^n_κ the subset of S^n of class $\kappa \in \mathcal{K}^n$. Then, for each $\kappa \in \mathcal{K}^n$, we compute the corresponding centroids as in Sec. 3.1:

$$c^n_\kappa = \frac{1}{|S^n_\kappa|} \sum_{i\in S^n_\kappa} \vec{x}_i. \tag{5}$$

Then, each centroid c^n_κ is assigned randomly to a left or right child node symbolized by a binary value $e_\kappa \in \{0,1\}$. The corresponding splitting function f is then defined by:

$$f(\vec{x}) = e_{\kappa^*(\vec{x})} \;\text{ where }\; \kappa^*(\vec{x}) = \operatorname*{argmin}_{\kappa\in\mathcal{K}^n} \left\|\vec{x} - c^n_\kappa\right\|^2. \tag{6}$$

We use Eq. (3) to select the optimal f^n from the pool of splitting functions corresponding to random centroids assignments $\{e_\kappa\}$. We do not optimize over random choices of $\mathcal{K}^n$ for two reasons. First, this would force us to store all class means at all nodes. Second, we can exploit reservoir sampling to add new classes to $\mathcal{K}^n$ in a principled manner. With $|\mathcal{K}^n| \ll |\mathcal{K}|$, the forests will perform a low number of the comparisons, and scale graciously as our experiments in Sec. 5 demonstrate.

The experiments further show that the proposed NCM splitting functions outperform standard ones for the task of large-scale image classification. We also show that the classification accuracy of NCM Forests without metric learning is comparable to the performance of NCM with metric learning, but the training of the random forest is intrinsically parallelizable and thus faster than the metric learning. Moreover, the main benefit of the approach is the ease of incrementally adding new classes to an already trained multiclass classifier as we discuss in the next section. Classification with a tree of an NCM Forest is illustrated in Fig. 1.

4. Incremental learning

As discussed in Sec. 2, on-line learning of random forests has been studied for vision applications such as tracking, object detection, or segmentation [11, 18, 25, 27]. However, these works focus on problems where the number of classes is known a-priori. In this work, we focus on incrementally adding new classes to the forest in the context of large scale image classification. Without proper incremental learning mechanism, a multi-class classifier would need to be re-trained from scratch every time a new class is added. This makes it potentially very expensive to add new classes, especially as the dataset grows. Below, we devise four approaches for incrementally learning an NCM Forest.

a) Update leaf statistics (ULS). Assuming that a multi-class NCM Forest has been already trained for the classes $\mathcal{K}$, a new class κ' is added by passing the training images of the new class through the trees and updating the class probabilities $P_l(\kappa)$ stored at the leaves. This approach updates only the leaves but does not change the splitting functions or size of the trees.

b) Incrementally grow tree (IGT). Unlike ULS, IGT continues growing the trees if enough samples of the new class arrive at a leaf. The previously learned splitting functions remain unchanged, but new splitting nodes can be added. While the newly added splitting functions sample centroids from $\mathcal{K} \cup \kappa'$, the old splitting functions are based on centroids sampled from $\mathcal{K}$.

c) Re-train subtree (RTST). In contrast to ULS and IGT, which do not converge to a forest trained on $\mathcal{K} \cup \kappa'$ classes (since the tree structure learned for $\mathcal{K}$ classes is not changed), RTST updates also previously learned splitting functions. To this end, a subset of nodes in the trees trained on $\mathcal{K}$ classes are marked and converted into leaves, *de facto* removing all of their children. By storing references to the training samples in leaves, it is efficient to collect them back for the newly created leaf node and update statistics. As for IGT, the cut trees are then grown again, which, in essence, corresponds to re-training subtrees with samples from all classes. The computational cost of re-training depends on the size of the sub-trees, so we set the probability of a node n to be marked as inversely proportional to the cardinality of the subtree T_n with n as root:

$$p(n) \propto (|T_n| + 1)^{-1}. \tag{7}$$

To control the amount of re-training, only a fraction $\pi \in [0, 1]$ of the subtrees is selected by randomly sampling without replacement. If $\pi = 1$, the trees are completely re-trained and the training is not incremental anymore. For $\pi = 0$, RTST is the same as IGT.

d) Re-use subtree (RUST). While RTST re-trains sub-trees entirely, we also propose a fourth approach that re-uses subtrees to reduce the training time. Instead of marking full sub-trees, RUST updates single splitting nodes. The nodes are selected for update as in RTST. The incremental training is then performed breadth-first. Each splitting node n already stores a function f^n where the $|\mathcal{K}^n|$ centroids have been sampled from $\mathcal{K}$. The splitting functions for $\mathcal{K} \cup \kappa'$ classes, however, would have been sampled from centroids from the larger set of classes. We therefore use reservoir sampling [24] to decide if the centroid $c^n_{\kappa'}$ is ignored, added or replaces an element of $\mathcal{K}^n$ to form $\mathcal{K}'^n$, in which case the splitting function is updated as well:

$$f'^n(\vec{x}) = e^n_{\kappa^*(\vec{x})} \text{ with } \kappa^*(\vec{x}) = \underset{\kappa \in \mathcal{K}'^n}{\operatorname{argmin}} \|\vec{x} - c^n_\kappa\|^2, \tag{8}$$

where $e^n_{\kappa'}$ is selected based on Eq. (3). Since updating the splitting function might result in a re-distribution of the training samples from the classes $\mathcal{K}$ within the sub-tree of the node, the samples with $f'^n(\vec{x}) \neq f^n(\vec{x})$ are removed from the leaves and passed through the subtree again. As this might create leaves without samples, the trees are cut such that each leaf contains a minimum number μ of samples. The impact of π and μ is evaluated in Sec. 5.

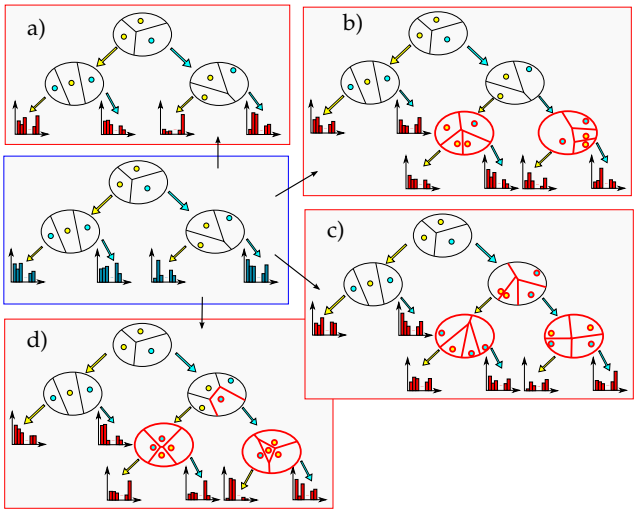
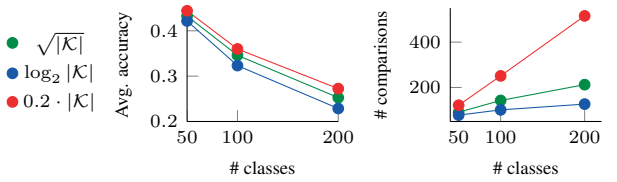


Figure 2: Illustration of our incremental approaches: **a)** Update leaf statistics (ULS), **b)** Incrementally grow tree (IGT), **c)** Re-train subtree (RTST), **d)** Re-use subtree (RUST). The colors of the centroids (yellow, cyan) indicate the directions associated with the Voronoi cells. The elements marked in red are modifications to the structure of the tree. In c), the centroids of the root's right child are re-computed, while in d) only a new centroid is added.

While ULS, IGT and RTST are general approaches that work with any type of splitting functions, RUST uses the advantage of NCM for incremental learning. Fig. 2 illustrates the four approaches for incremental learning.

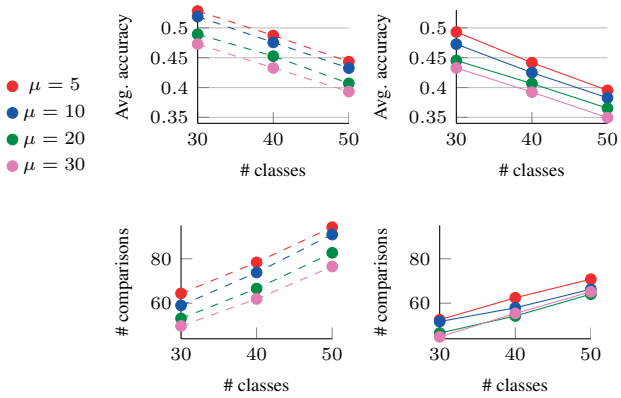
5. Experiments

For evaluation, we use the challenging ImageNet Large Scale Visual Recognition 2010 challenge benchmark (ILSVRC10) [3]. There are 1k categories (between 660 and 3047 training images per class, 1.2M in total; 150 testing images per category) organized in a class hierarchy based on WordNet. The performance is measured using average accuracy, *i.e.* the fraction of test samples correctly classified. We used densely sampled SIFT features clustered into 1k visual words provided by [3]. Although more advanced features [13, 20] improve results, the design and evaluation of feature sets is beyond the scope of this work. To evaluate incremental learning, we fixed a random order of all categories and used it throughout all experiments. Each feature was whitened by its mean and standard deviation over the starting training subset. Test time of our forests is measured as average number of comparisons $\|\vec{x} - c_\kappa\|^2$ per tree.



a) performance b) test time

Figure 3: Comparison of **a)** average classification accuracy and **b)** test time for different sizes of $\mathcal{K}^n \subset \mathcal{K}$. While setting $|\mathcal{K}^n|$ linear to the number of classes performs better than a square root or logarithmic growth, it takes much longer at the test time.



a) NCM Forest b) RUST

Figure 4: The influence of the μ minimum samples at a leaf for **a)** NCM forest and **b)** RUST with 3 initial classes.

5.1. Parameters of NCM Forest

We first evaluate the impact of some of the parameters of the NCM Forests. Unless stated otherwise, we used 50 trees, sampled 1024 functions at each node without replacement (*i.e.*, $|\mathcal{F}_n| = 1024$) and enforced at least $\mu = 10$ training samples at a leaf. One parameter of f^n, *cf*. Eq. (6), is the size of the sampled classes $\mathcal{K}^n$ out of all classes $\mathcal{K}$. We compared $|\mathcal{K}^n| \in \left\{\log |\mathcal{K}|, \sqrt{|\mathcal{K}|}, 0.2|\mathcal{K}|\right\}$ and present the results in Fig. 3. The standard deviation of the test time was $< 10\%$ of the mean and was omitted for clarity. The results show that $|\mathcal{K}_n| = \sqrt{|\mathcal{K}|}$ gives a good trade-off between accuracy and test time and is used for the rest of the paper.

The minimum number of samples at a leaf μ defines the stopping criterion for growing the trees. The smaller the number, the deeper the trees grow. Fig. 4a) shows that a small number increases the accuracy, but induces more comparisons at the test time. For the following experiments, we constrained leaves to contain at least 10 samples.

5.2. Comparison to other methods

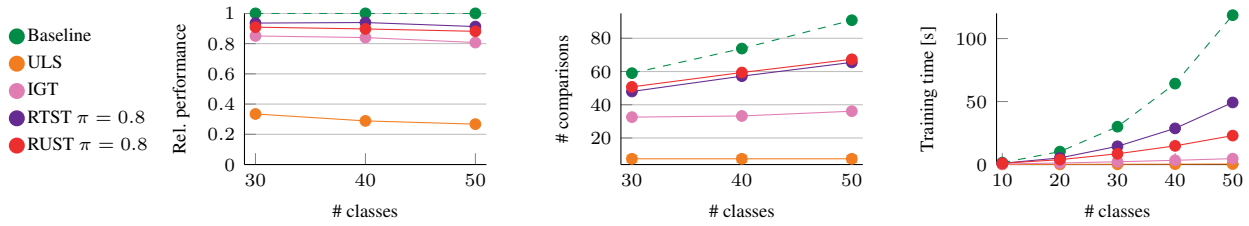
We compared NCM Forests with other multi-class classifiers using the same features. For comparison, we used nearest class mean classifier (NCM), NCM with metric learning [14] (MET+NCM), structured-output multi-class SVM [1] (MC SVM), k-nearest neighbors (KNN), and Random Forests with axis-aligned splitting functions [5] (RF), which performed better than RF with random linear splitting functions in our case. For each approach, the parameters were optimized by cross-validation for the first 50 classes. The results in Fig. 10a) show that NCM Forests perform comparable to NCM with metric learning [14] and outperform the other methods. In particular, NCMF outperforms NCM (Sec. 3.1) and conventional Random Forests (Sec. 3.2), by a margin of at least 10 points. The performance drop of MC SVM for more than 200 classes is likely due to the sensitivity of the parameters optimized for 50 classes.

5.3. Incremental learning

To evaluate the incremental learning approaches presented in Sec. 4, we train an NCMF on a pre-defined number k of initial classes and then incrementally add the other classes one by one. The performance is measured whenever the method has learned to recognize a certain number of classes. Since the goal is to match the performance of the NCMF re-trained at every new class, we measure the performance relatively to that baseline. The training time is measured in seconds of wall time.

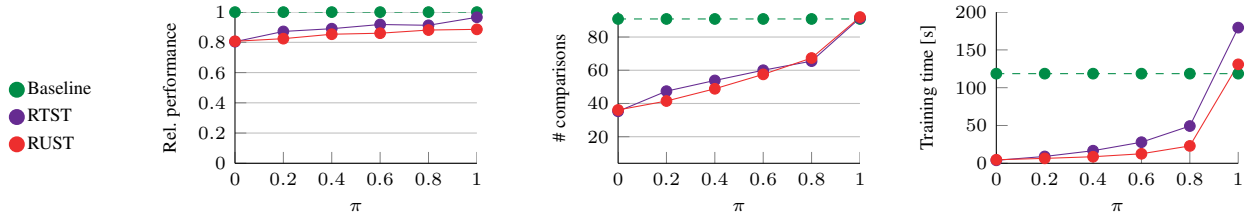
Comparison of the update strategies. Fig. 5 plots the accuracy, test time, and training time for the baseline and our approaches 'Update leaf statistics' (ULS), 'Incrementally grow tree' (IGT), 'Re-train subtree' (RTST with $\pi = 0.8$) and 'Re-use subtree' (RUST with $\pi = 0.8$) trained from $k = 3$ initial classes up to 50 classes. As ULS does not grow the trees, its test time is constant and the training time very low, but the final relative performance at 50 classes is poor (26.7%). IGT extends the trees, yielding higher test and training times, but achieves 80.7% relative performance while reducing training time of the baseline by a factor of 25 and test time by 3. IGT achieves 34.9% average accuracy, outperforming NCM, KNN, and RF, *cf*. Fig. 10a). RTST retrains the nodes and achieves the best relative performance (91.2%), but takes longest to train. RUST outperforms IGT (88.1% relative performance), suggesting that re-using the subtrees is indeed beneficial. It also speeds up the training of the baseline by a factor of 5 and is 2x faster to train than RTST. The gap in training times between RTST and RUST widens with the number of classes.

Impact of π. RTST and RUST depend on the parameter π, whose impact is shown in Fig. 6. If $\pi = 0$, RTST and RUST are the same as IGT. As π increases, RTST converges to the baseline. Although RTST with $\pi = 1$ is theoretically



a) relative performance b) test time c) training time

Figure 5: Measurements at variable number of classes starting with 3 initial classes. 'Update leaf statistics' (ULS) is faster to train and test, but has inferior performance. 'Incrementally grow tree' (IGT) is slower than ULS both at train and test time, but achieves 80.7% of the baseline's performance at 50 classes. 'Re-train subtree' achieves the best performance (91.2% at 50 classes), but takes longest to train. 'Re-use subtree' (RUST) is a good trade-off between training time and relative performance (88.1% at 50 classes). The relative differences in training time increase with the growing number of classes.



a) relative performance b) test time c) training time

Figure 6: The influence of π on 'Re-train subtree' (RTST) and 'Re-use subtree' (RUST) on **a)** performance, **b)** test time and **c)** training time. We started with 3 initial classes and measured at 50 classes. As π increases, the performance tends to the baseline at cost of higher training time. $\pi = 0.8$ appears as a good trade-off for both RTST and RUST.

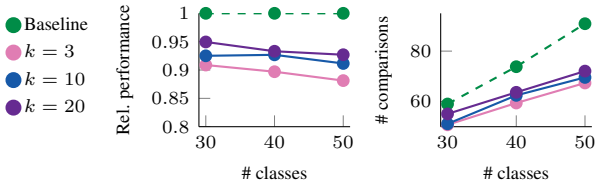
the same as the baseline, the sampling of the node for re-training creates an additional overhead. The performance is also slightly below the baseline because the whitening parameters are estimated on the initial training set, while the baseline computes the feature normalization based on all the data.

The impact of π on RUST is similar to RTST, but RUST does not converge to the baseline. This is expected since the stored centroids are re-used for RUST while re-computed for RTST. This results in a slightly lower relative performance than RTST, but also in lower training times.

Other parameters. We now focus on RUST and evaluate the impact of different parameters for incremental learning. As in Sec. 5.1, we evaluated the impact of the stopping criterion given by the minimum number of samples at a leaf. Fig. 4b) shows that the impact is also limited for RUST.

The influence of the number of initial classes on RUST is shown in Fig. 7. The method is quite insensitive to the number of initial classes and already achieves good performance with only a few. Starting with 3 and 20 initial classes gives us the relative performance of 88.1% and 92.7%, respectively, a difference of only about 4.5%.

So far we have used a single random permutation of the classes for the experiments. Below, we evaluate ten random permutations of the previously used 50 classes, to ensure that the specific order that we have used does not bias our



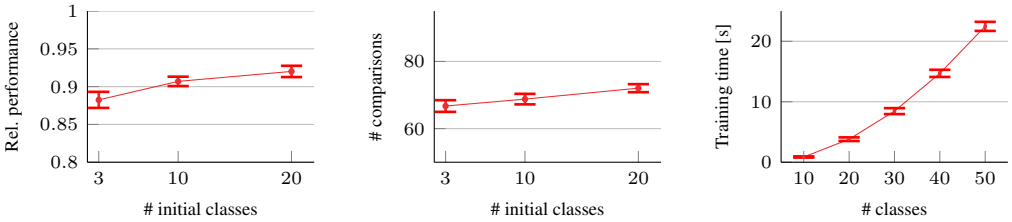
a) relative performance b) test time

Figure 7: Comparison of **a)** relative performance and **b)** test time of RUST with $\pi = 0.8$ starting with k initial classes measured at 30, 40 and 50 classes. Increasing the number of initial classes is beneficial, but has limited impact.

evaluation. The standard deviation never exceeded 10% of the mean values of the measurements (*cf*. Fig. 8) indicating little impact of the order of the classes, which is desirable for incremental learning.

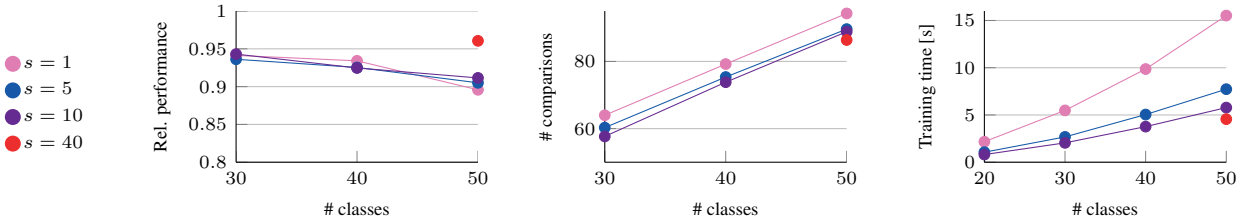
In practice, multiple classes can appear in batches, so the number of simultaneous classes to add is an interesting parameter to study. We initialized with 10 classes and experimented with step chunks of 1, 5, 10 or all classes (40). While only large chunks increase the accuracy, the training time is already reduced by training 5 classes at a time as shown in Fig. 9.

While we compared NCM Forests with other approaches in Fig. 10a), we now compare the incremental learning approaches on all 1k classes in Fig. 10b) and c). Since IGT al-



a) relative performance b) test time c) training time

Figure 8: Comparison of **a)** relative performance and **b)** test time of RUST with $\pi = 0.8$ starting with different number of classes and measured at 50 classes and 10 random permutations of the classes. **c)** Training time for 3 initial classes over 10 random permutations of the classes. The small standard deviations indicate the limited impact of the order of the classes.



a) relative performance b) test time c) training time

Figure 9: Comparison of **a)** relative performance, **b)** test time and **c)** training time for RUST when several classes (s is the chunk size) are added simultaneously, starting from 10 initial classes.

ready outperforms NCM, KNN, and RF, we focus on NCM with metric learning [14], which performed slightly better than NCM Forests, *cf*. Fig. 10a). We start with $k = 10$ and $k = 20$ initial classes. The setting for the approaches based on NCM Forests remains the same, *i.e.*, the whitening is estimated on the initial k classes. For MET$_k$+NCM, the metric is only learned on the initial classes, and the model is updated with projected centroids of the new classes. In Fig. 10, RUST outperforms IGT showing that updating the trees is beneficial. While it was shown in [14] that a metric learned on 800 classes is applicable to the other 200 classes, the learned metric on up to 20 classes does not generalize well, making the method unsuitable for a small initial training set. In this case, the three approaches IGT, RUST and RTST outperform MET$_k$+NCM. In different scenarios, better accuracy (RTST) or faster training (RUST) may be preferred, *cf*. Fig. 5.

Finally, we report the total training times to reach 1k classes for a tree initially trained on 20 classes on a single-threaded machine: re-training NCMF baseline for each new class takes 4 days, ULS 30s, IGT 15min, RUST 77min and RTST 16h, respectively. Without feature extraction, a tree processes an image in: NCMF 0.47ms, ULS 0.04ms, IGT 0.05ms, RUST 0.24ms and RTST 0.27ms, respectively.

6. Conclusion

In this paper, we have introduced Nearest Class Mean Forests (NCMF), and shown that they outperform NCM classification and RFs for large-scale image classification. While the approach achieves competitive results in a setting where all classes are known a-priori, efficient techniques to incrementally add new classes to NCMF are also proposed. In particular, the ability to re-use subtrees allows us to add new classes at a fraction of the cost of re-training a complete NCMF, while preserving the overall accuracy. We have performed extensive experiments in the context of classification when the number of classes grows over time. Since NCMF are quite insensitive to the number of initial classes and to the order in which the classes are added, they are well suited for incremental learning.

Acknowledgements: The research was supported in part by the DFG Emmy Noether program (GA 1927/1-1), Swiss CTI (15769.1) and Toyota.

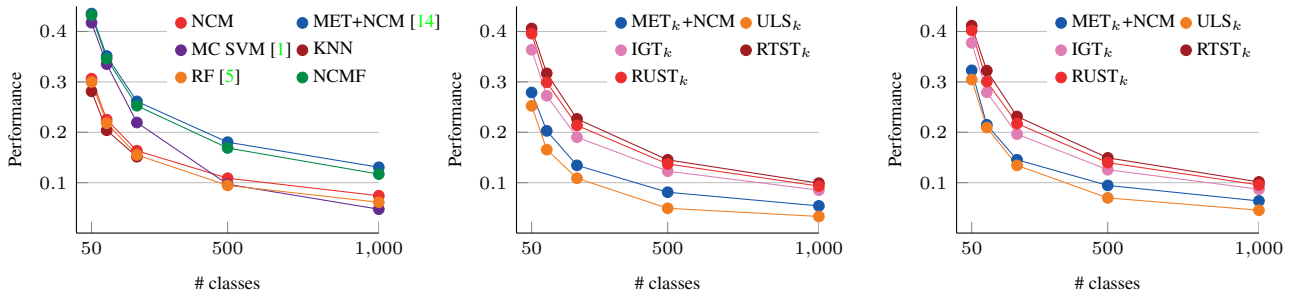
References

[1] Z. Akata, F. Perronnin, Z. Harchaoui, and C. Schmid. Good Practice in Large-Scale Learning for Image Classification. *TPAMI*, 2013.

[2] Y. Aytar and A. Zisserman. Tabula rasa: Model transfer for object category detection. In *ICCV*, 2011.

[3] A. Berg, J. Deng, and L. Fei-Fei. Large scale visual recognition challenge 2010. http://www.image-net.org/challenges/LSVRC/2010, 2010. [Online; accessed 1-Nov.-2013].

[4] A. Bosch, A. Zisserman, and X. Muñoz. Image classification using random forests and ferns. In *ICCV*, 2007.



a) baseline b) $k = 10$ initial classes c) $k = 20$ initial classes

Figure 10: Comparison of baselines and different incremental learning methods on 50, 100, 200, 500 and 1k classes of [3] all starting with the same initial classes. The whitening for our methods as well as the metric MET$_k$ were learned on k initial classes in b) & c). We set $\pi = 0.8$ for RUST and RTST. RUST and RTST consistently outperform the competing incremental methods.

[5] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[6] J. Deng, A. C. Berg, K. Li, and L. Fei-Fei. What does classifying more than 10,000 image categories tell us? In *ECCV*, 2010.

[7] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR*, 2009.

[8] J. Deng, J. Krause, A. Berg, and L. Fei-Fei. Hedging your bets: Optimizing accuracy-specificity trade-offs in large scale visual recognition. In *CVPR*, 2012.

[9] P. Domingos and G. Hulten. Mining high-speed data streams. In *SIGKDD*, 2000.

[10] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *IJCV*, 2010.

[11] M. Godec, P. Roth, and H. Bischof. Hough-based tracking of non-rigid objects. In *ICCV*, 2011.

[12] M. Guillaumin and V. Ferrari. Large-scale knowledge transfer for object localization in imagenet. In *CVPR*, 2012.

[13] Y. Lin, F. Lv, S. Zhu, M. Yang, T. Cour, K. Yu, L. Cao, and T. Huang. Large-scale image classification: Fast feature extraction and svm training. In *CVPR*, 2011.

[14] T. Mensink, J. Verbeek, F. Perronnin, and G. Csurka. Distance-based image classification: Generalizing to new classes at near-zero cost. *TPAMI*, 2013.

[15] D. Nistér and H. Stewénius. Scalable recognition with a vocabulary tree. In *CVPR*, 2006.

[16] P. Ott and M. Everingham. Shared parts for deformable part-based models. In *CVPR*, 2011.

[17] M. Rohrbach, M. Stark, and B. Schiele. Evaluating knowledge transfer and zero-shot learning in a large-scale setting. In *CVPR*, 2011.

[18] A. Saffari, C. Leistner, J. Santner, M. Godec, and H. Bischof. On-line random forests. In *OLCV*, 2009.

[19] R. Salakhutdinov, A. Torralba, and J. Tenenbaum. Learning to share visual appearance for multiclass object detection. In *CVPR*, 2011.

[20] J. Sanchez and F. Perronnin. High-dimensional signature compression for large-scale image classification. In *CVPR*, 2011.

[21] M. Stark, M. Goesele, and B. Schiele. A shape-based object class model for knowledge transfer. In *ICCV*, 2009.

[22] T. Tommasi, F. Orabona, and B. Caputo. Safety in numbers: Learning categories from few examples with multi model knowledge transfer. In *CVPR*, 2010.

[23] A. Torralba, R. Fergus, and W. T. Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *TPAMI*, 30(11):1958–1970, 2008.

[24] J. S. Vitter. Random sampling with a reservoir. *ACM Trans. Math. Softw.*, 1985.

[25] A. Wang, G. Wan, Z. Cheng, and S. Li. An incremental extremely random forest classifier for online learning and tracking. In *ICIP*, 2009.

[26] J. Xiao, J. Hays, K. A. Ehinger, A. Oliva, and A. Torralba. Sun database: Large-scale scene recognition from abbey to zoo. In *CVPR*, 2010.

[27] A. Yao, J. Gall, C. Leistner, and L. Van Gool. Interactive object detection. In *CVPR*, 2012.

[28] B. Yao, A. Khosla, and L. Fei-fei. Combining randomization and discrimination for fine-grained image categorization. In *CVPR*, 2011.

[29] T. Yeh, J. Lee, and T. Darrell. Adaptive vocabulary forests br dynamic indexing and category learning. In *ICCV*, 2007.